Perspective: Availability

CSCE 742 - Lecture 19 - 11/15/2018

Availability

- Is the software there and ready to carry out its task when you need it to be?
  - Encompasses **reliability** and **repair**.
  - Reliability: Does the system tend to demonstrate correct behavior?
  - Repair: Can the system recover from an error?
- **Availability** refers to the ability of a system to mask or repair faults such that the cumulative service outage does not exceed a required value over a time interval.

Availability

- Closely related to security and performance.
  - Security: A DDOS attack is designed to make the system unavailable.
  - Performance: Has the system failed, or is it recovering or limiting the damage from a hazard?
- Availability is about minimizing outage time by mitigating faults.
  - A **failure** is a visible deviation from expected behavior (crash, incorrect output).
  - Failures are caused by **faults** - a mistake in the source code.

Availability

- Achieving availability requires understanding the nature of the failures that can arise.
- Faults and failures can be prevented, tolerated, removed, or forecasted.
  - How are faults detected?
  - How frequently do failures occur?
  - What happens when a failure occurs?
  - How long can the system be out of operation?
  - When can faults or failures occur safely?
  - Can faults or failures be prevented?
  - What notifications are required when failure occurs?

Measuring Availability

- Time to repair is the time until failure is no longer observable.
- “Observability” can be hard to define.
  - Stuxnet caused problems for months before being noticed. How does that impact availability?
- Software can remain partially available more easily than hardware.
  - “Degraded operating modes”
- If code containing a fault is executed, but the system is able to recover, there was no failure.

Measuring Availability

- Availability is the probability that the system will provide a service within required bounds over a specified time interval.
  - **Availability = MTBF / (MTBF + MTTR)**
    - MTBF: Mean time between failures.
    - MTTR: Mean time to repair
- Scheduled downtime often does not count.
- Pay attention to the significant figures!
  - 0.99999 = down for 5 minutes per year **(high avail.)**
  - 0.9999 = down for 52 minutes per year
  - 0.99 = down for 3 days per year

Impact on Views

- Context:
  - Consider how your system's availability is impacted by availability of external systems.
- Functional:
  - Availability is a key concern to user and acquirer stakeholders.
  - Functional changes may be needed to support availability requirements, such as the ability to operate in an offline mode when a network is unavailable.

Impact on Views

- Information:
  - Consider the set of processes and systems for backup and recovery.
    - Systems must be backed up in such a way that they can be recovered in a reasonable amount of time if a disaster occurs.
    - Backups should not impact online availability, or if they do, they may need to be scheduled to occur outside the online day
- Concurrency:
  - Features such as hardware replication and failover may imply changes to your concurrency model.

Impact on Views

- Development:
  - Achieving availability may impose constraints on software development.
  - All elements may have to support start, stop, pause, and restart commands.
- Deployment:
  - Availability has major impact on deployment.
  - Requirements may mandate environment where hardware is duplicated or a recovery site that can be activated if production environment goes down.
  - May need special software to support hardware redundancy and clustering.

Impact on Views

- Operational:
  - Processes to identify and recover from problems may be required.
  - May be a need for geographically-separate disaster recovery facilities.
  - Processes for main site failover, network failover, data recovery must be designed, tested, and built.
  - If standby site is physically separate from production site, processes are needed to move staff between locations.

Planning for Failure

“Failure is not an Option”

- **Failure is absolutely an option.**
  - Failure is inevitable.
- Making a system safe and available requires planning for and handling failures.
  - First step - understanding what kinds of failures the system is prone to and the consequences.
- Techniques:
  - Hazard Analysis
  - Fault Tree Analysis

Hazard Analysis

- Catalog the hazards that can occur, classified by severity.
- Example from Aeronautics:
  - Catastrophic: Represents the loss of critical function required to safely fly and land aircraft.
  - Hazardous: Failure has a large negative impact on safety, performance, or ability to operate the aircraft.
  - Major: Leads to passenger discomfort or increases crew workload to the point where safety is affected.
  - Minor: Causes passenger inconvenience or a routine flight plan change.
  - No Effect: No impact on safety, crew, or operation.

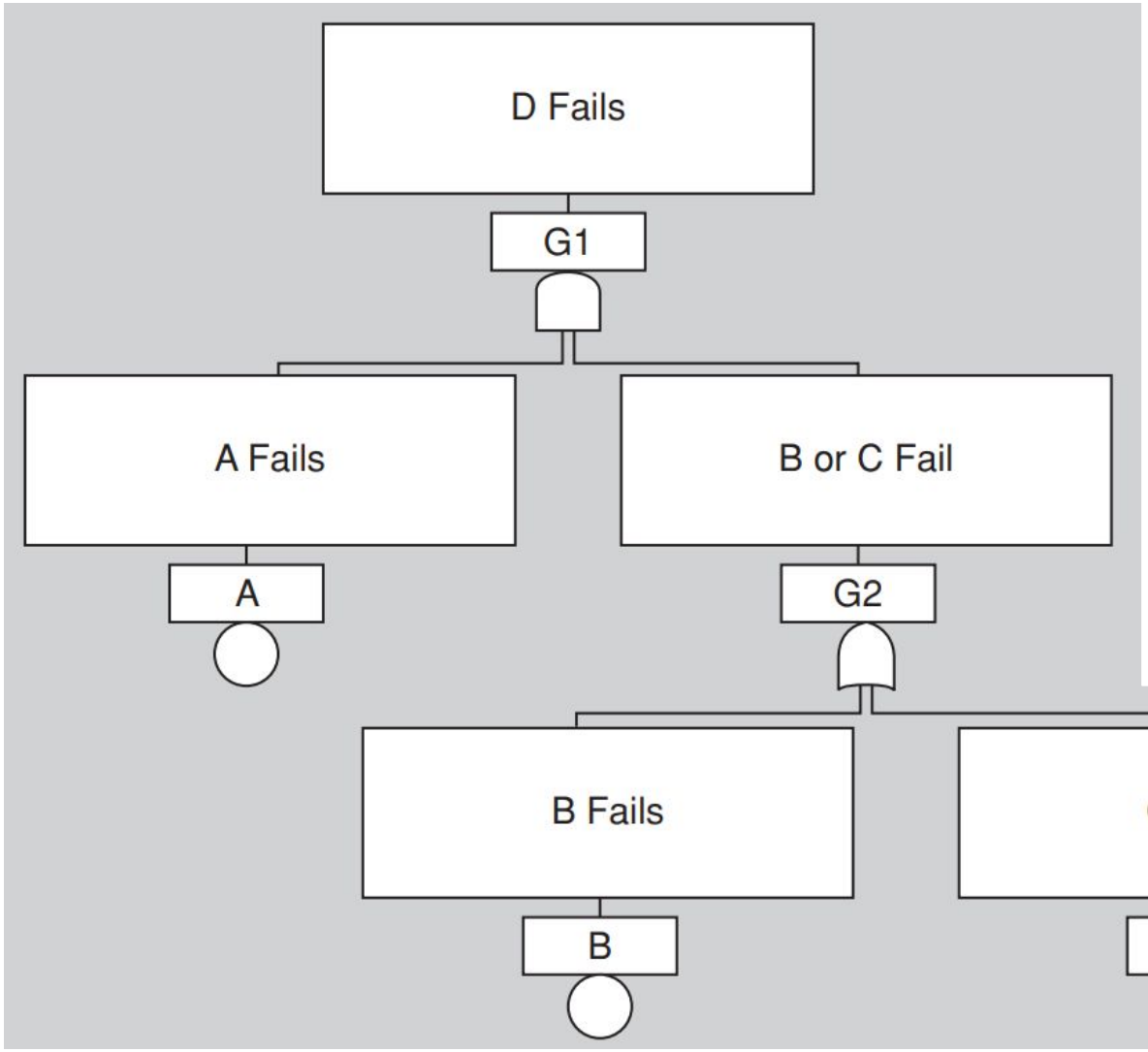
Hazard Analysis

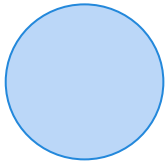
- Requires assessment of probability a hazard will occur.
  - If (cost * probability) > threshold, we must develop a way to mitigate the hazard.
- Common Thresholds
  - **Probable:** Probability of occurrence per operational hour is greater than $1 * 10^{-5}$
  - **Remote:** Probability of occurrence per operational hour is less than $1 * 10^{-5}$ and greater than $1 * 10^{-7}$
  - **Extremely Remote:** Probability per operational hour is less than $1 * 10^{-7}$ and greater than $1 * 10^{-9}$
  - **Extremely Improbable:** Probability of occurrence per operational hour is less than $1 * 10^{-9}$

Fault Tree Analysis

- Specifies a state of the system that impacts safety or reliability, then analyzes operation to find all ways that the state could occur.
- Fault tree identifies all sequential and parallel sequences of contributing faults that could result in the failure.
  - Can be hardware, human, or software faults.
  - Can be events that drive the system out of normal operating conditions.

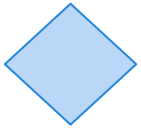
Fault Tree Notation





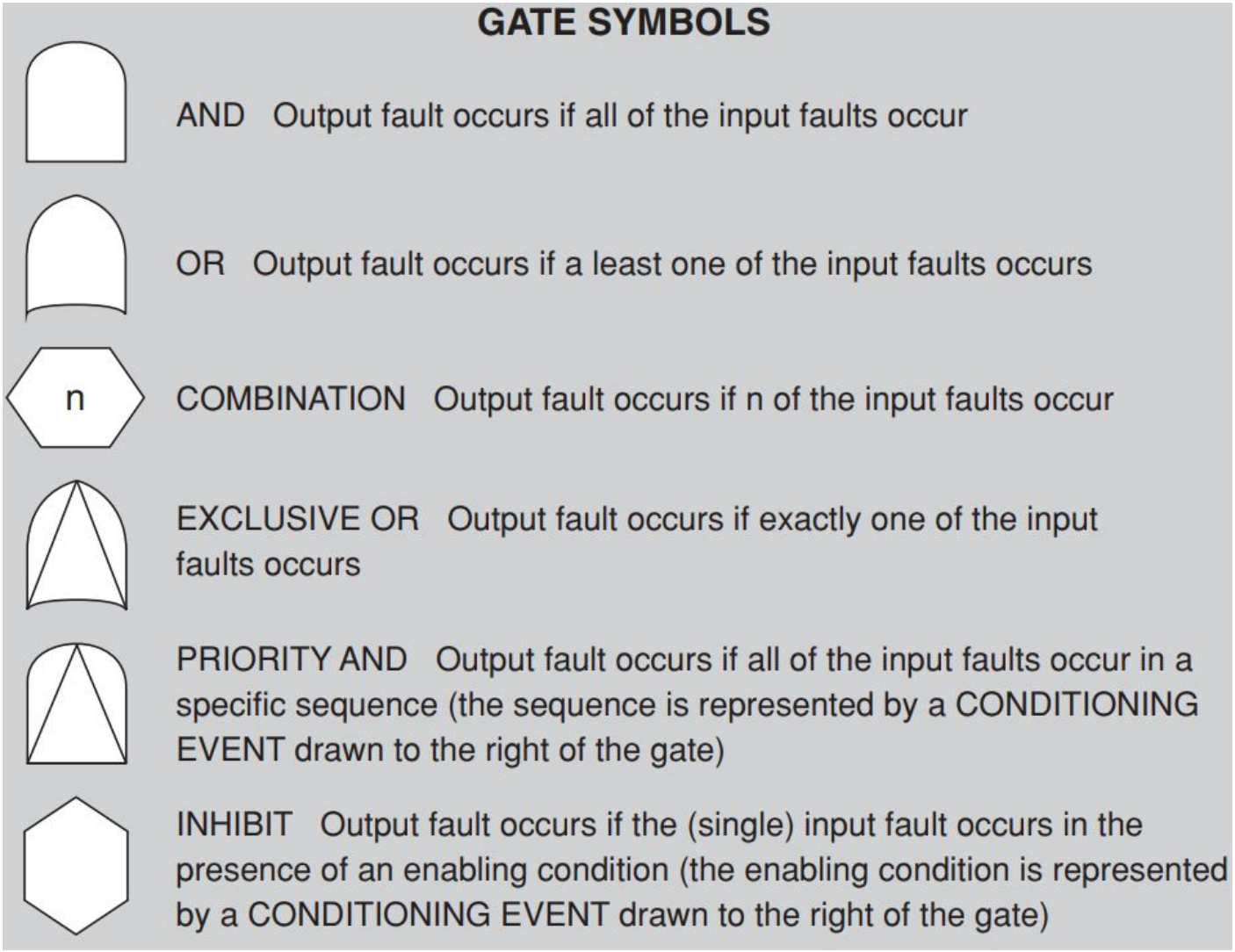
Basic Event (generally an environmental factor or input)

Intermediate Event (generally a failure)

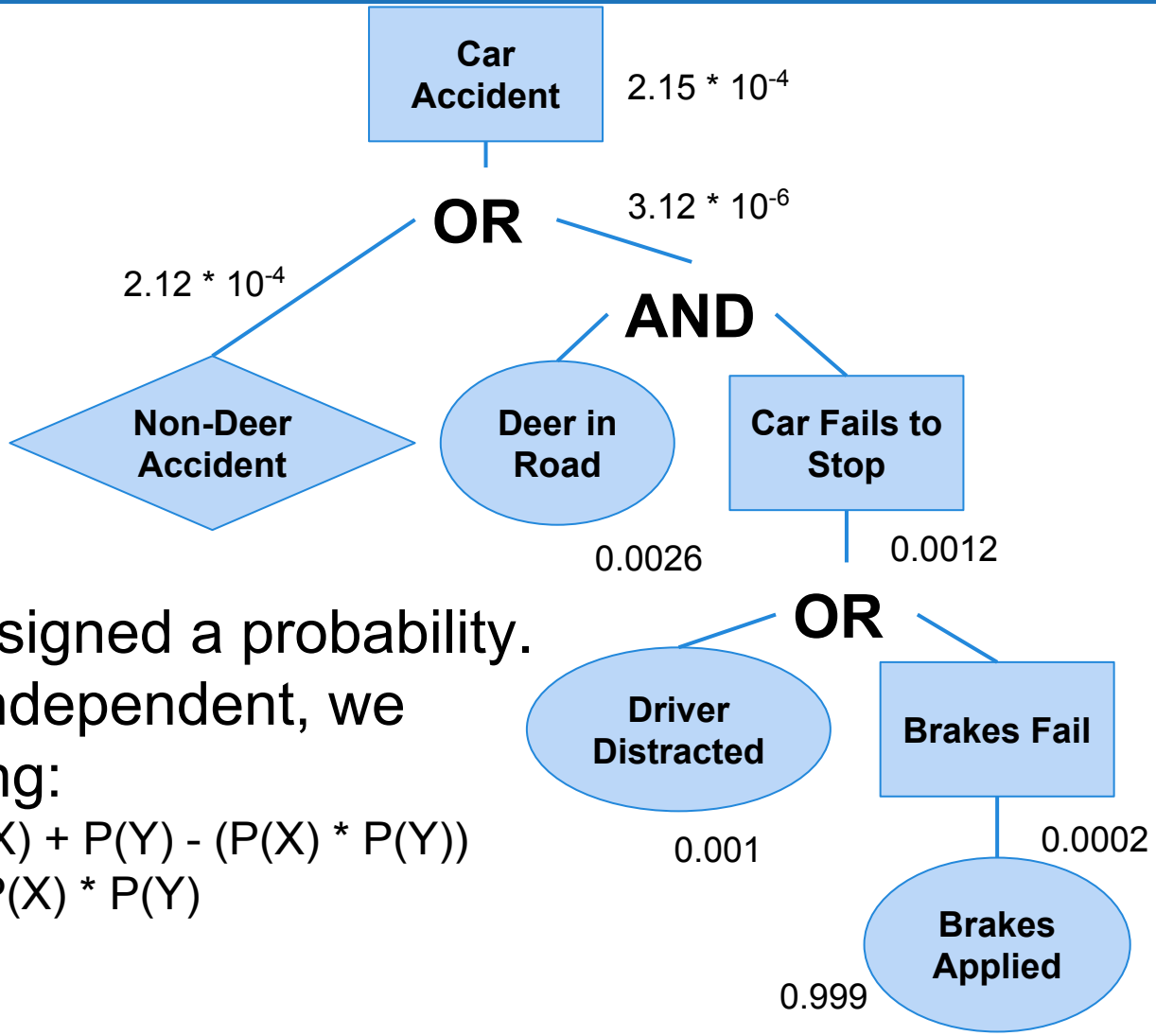


Undeveloped Event (we are not modeling it or digging into it)

Fault Tree Notation



Fault Tree Example



- Each event assigned a probability.
- If events are independent, we propagate using:
  - P(X or Y) = P(X) + P(Y) - (P(X) * P(Y))
  - P(X and Y) = P(X) * P(Y)

Availability Scenarios

Availability Quality Scenarios

- The ability of the system to mask or repair faults such that the outage period does not exceed a required value over a time period.
- Measure how the system responds to failure.
  - When the system breaks, how long does it take to resume normal operation?
- Stimuli should always be a failure.
- Response measures should always include a measure of availability:
  - availability percentage, time to detect or repair fault, time system in degraded mode, no down time, etc.

Availability Quality Scenarios

- Scenarios must distinguish physical failures in the system and the software's perception of the failure.
  - Do not assume software is omniscient.
- Scenarios tend to deal with:
  - Failure of a physical component or external system.
  - Reconfiguration of the physical system.
  - Maintenance or reconfiguration of the software.

Generic Availability Scenario

- **Overview:** Description of the scenario.
- **System/environment state:** The state of the system when the fault or failure occurs may also affect the desired system response. If the system has already failed and is not in normal mode, it may be desirable to shut it down. If this is the first failure, degradation of response time or functions may be preferred.
- **External Stimulus:** Differentiate between internal and external origins of failure because desired system response may be different. Stimuli is an *omission* (a component fails to respond to an input), a *crash* (component repeatedly suffers omission faults), *timing* (a component responds but the response is early or late) or *response* (a component responds with an incorrect value).

Generic Availability Scenario

- **Required system behavior:** There are a number of possible reactions to a failure. Fault must be detected and isolated before any other response is possible. After the fault is detected, the system must recover from it. Actions include logging the failure, notifying selected users or other systems, taking actions to limit the damage caused by the fault, switching to a degraded mode with either less capacity or less function, shutting down external systems, or becoming unavailable during repair.
- **Response measure:** Can specify an availability percentage, or it can specify a time to detect the fault, time to repair the fault, times or time intervals where system must be available, or duration for which the system must be available.

Availability Scenario

Availability while adding new taps

- **Overview:** How the system handles additional taps being added to the system.
- **System/environment state:** The system is operating normally, without problems.
- **External Stimulus:** A user powers up a new Kegboard on the network with six additional taps.
- **Required system behavior:** The kegboards send init messages to the central Kegbot server. The server interrogates the kegboards and adds the additional taps to the inventory of taps. The system continues to service the existing taps without interruption.
- **Response measure:** There is no interruption of service to existing taps. Within 1 second, the new kegboard is added to the administrative interface on the KegBot web server for administraton configuration.

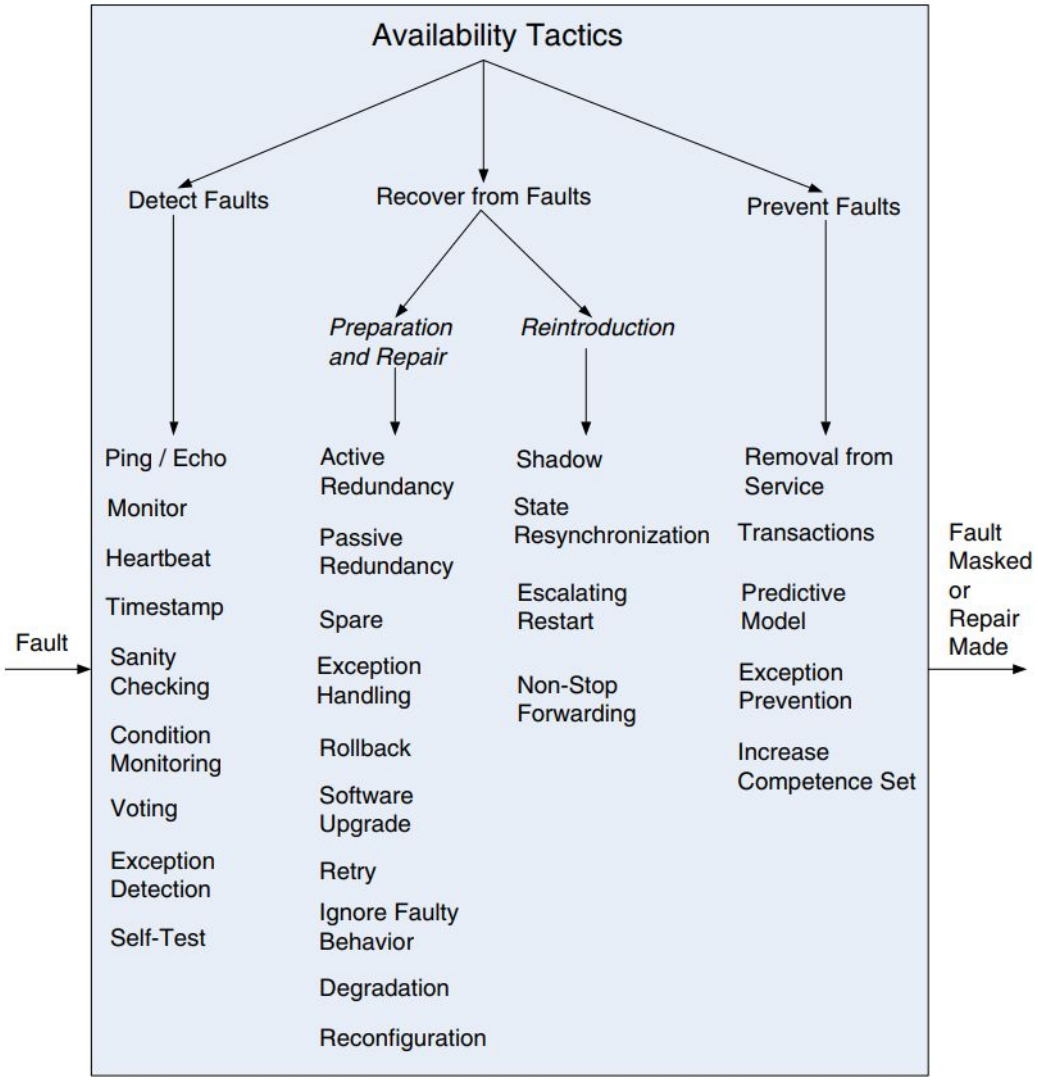
Availability Scenario 2

Web server failure at e-commerce site

- **Overview:** One of the client-facing web servers fails during transmission of client page update.
- **System/environment state:** System is working correctly under normal load. Customer has generated a “add item to shopping cart” post, which was routed to web server <X> in transaction pool.
- **External Stimulus:** Web server <X> crashes during response generation.
- **Required system behavior:** Response page may be corrupted on client browser. Load balancer component no longer receives heartbeat message from web server and so removes it from the pool of available servers after 2s of missed messages, or upon next request sent to the server. Load balancer will remove the server from the pool of available servers. From client’s perspective, a page reload will be automatically routed to alternate server by load balancer and page will be correctly displayed.
- **Response measure:** Upon client-side page refresh, client state and display contains state after last transaction. Time for re-routed refresh is equivalent to “standard” refresh (<1 second 95% of the time).

Availability Tactics

Availability Tactics



Detect Faults

- Before we can take action, the presence of a fault must be detected or anticipated.
- Ping/Echo:
  - Asynchronous request/response message exchanged between nodes to determine reachability and round-trip delay.
  - Echo determines that a pinged element is alive.
  - Ping often sent be a system monitor.
  - Time threshold determines how long to wait for the echo before timing out.
  - Standard implementations available through IP.

Detect Faults

- Monitor:
  - Element that monitors the health of processes, CPUs, I/O, memory, etc.
  - Can detect failure or congestion in network and shared resources,initiate system self-tests, and detect timing issues.
    - “Watchdog” that monitors overall system.
- Heartbeat:
  - Message exchange between monitor and a process.
  - Process resets watchdog timer in monitor.
  - Often merged with other control messages to avoid performance or scalability issues.
  - Difference with ping: process has the responsibility.

Detect Faults

- Time Stamp:
  - Detect incorrect sequences of events.
  - Time stamps established using local clocks.
  - Can use a sequence number of time is not important
- Sanity Check/Condition Monitoring:
  - Checks validity or reasonableness of specific operations or outputs of a component.
  - Requires well-specified properties.
  - Employed at interfaces to examine information flow.
  - Monitor must be simple to avoid introducing new errors through the overhead of monitoring.

Detect Faults

- Voting:
  - Run three elements that “do the same job”, apply same input to each, then compare the output.
    - Often three copies of the same process.
  - Detect inconsistency among the three output states.
  - If there are disagreements, report a fault.
  - Use majority as the “official” output.

Failure Recovery

- Prevention-and-repair tactics attempt to retry computations or introduce redundancy.
- Active Redundancy (Hot Spare):
  - A protection group is a group of processing nodes where one or more nodes are “active,” with the remaining nodes serving as redundant spares.
  - In this configuration, all nodes (active or redundant) process input in parallel, allowing redundant spares to synchronize with active nodes.
  - Because redundant nodes has identical state, it can take over from a failed element in milliseconds.

Failure Recovery

- Passive Redundancy (Warm Spare):
  - Only active members of a production group process input, and provide redundant spares with status updates.
  - State of redundant nodes is only loosely coupled to active nodes.
  - Enables availability with lower performance cost.
- Spare (Cold Spare):
  - Redundant nodes remain out of service until a failure occurs, then they replace the active nodes.
  - Poor recovery performance, but improves reliability.

Failure Recovery

- Exception Handling:
  - Once an exception is detected, the system must handle it.
  - Easiest thing to do is crash, but this is not desirable.
  - Offer error messages, details on why the program failed (cause and location of the failure).
  - Mask the fault and restore the program to a usable state or retry an operation.
- Rollback:
  - Revert to a known good state (“rollback line”).
  - Continue from restored state.
  - Often paired with redundancy.
  - Can update stored state when convenient.

Failure Recovery

- Retry:
  - Assumes fault is transient. Retries the failed operation. Often used in networked environments.
  - Limit number of retries.
- Ignore Faulty Behavior:
  - Block all messages from a source known to be faulty
  - Common tactic for avoiding DDOS attacks.
- Degradation:
  - Drop non-critical functions after a failure and maintain the functioning critical services.
- Reconfiguration:
  - Reassign responsibilities to remaining resources.

Failure Recovery

- Reintroduction restores failed elements after correction.
- Shadow:
  - Operate a failed or in-service component in a “shadow” mode for a period of time before restoring to active service.
  - During this time, its behavior is monitored for correctness and state is re-populated incrementally.
- State Resynchronization:
  - With active redundancy: States of active and passive elements are compared to ensure synchronization.
  - With passive redundancy: State of passive elements is periodically updated by active elements.

Failure Recovery

- Escalating Restart:
  - Allows the system to recover from failures by varying the granularity of the elements restarted and minimizing level of service affected.
  - Level 0: Employ a warm spare. All child threads of the faulty element are killed and recreated.
  - Level 1: Frees and reinitializes all unprotected memory.
  - Level 2: Frees and reinitializes all memory, forcing all elements to reload and reinitialize.
  - Level 3: Completely reload and reinitialize the executable image and associated data.
  - Useful for enabling graceful degradation.

Failure Prevention

- Removal from Service:
  - Temporarily place elements in out-of-service states to mitigate potential system failures.
  - Take an element out of service and reset it to scrub latent faults (memory leaks, fragmentation, caching errors) before they cause a failure.
- Transactions:
  - Ensure that all asynchronous messages are:
    - Atomic (each transaction treated as a single unit, and either succeeds or fails completely)
    - Consistent (data changes must be valid)
    - Isolated (concurrent updates have same effect as sequential)
    - Durable (once completed, a transaction remains committed even in case of a failure).

Failure Prevention

- Transactions:
  - Usually implemented through two-phase commit.
    - All processes in the transaction asked whether to commit or abort the transaction.
    - Then, a manager decided whether to commit based on process votes.
  - Prevents race conditions from processes attempting to update at the same time.

Failure Prevention

- Predictive Model
  - A monitor uses a model to predict when the system has stepped outside of normal operating parameters, and takes action.
  - Uses performance metrics to predict failure.
    - Session establishment rate (for a server).
    - Process state (in-service, out-of-service, under maintenance, idle).
    - Message queue length.
    - Task completion speed.

Failure Prevention

- Exception Prevention
  - Prevent exceptions from occuring.
  - Use “wrappers” around common datatypes to prevent misuse of that type.
    - Prevent dangling pointers, semaphore access violations, null pointers, out-of-bounds access.
  - Use smart pointers to do bounds-checking on pointers and ensure that resources are automatically deallocated when no data refers to it.
    - Avoids resource leaks.

Availability Design Guidelines

Allocation of Responsibilities

- Determine system responsibilities that need to be highly available.
- For each, ensure that the system can:
  - Detect an omission, crash, incorrect timing, or incorrect response.
  - Log the failure and details on underlying fault.
  - Notify appropriate people or systems.
  - Disable the source of events causing the failure.
  - Be temporarily unavailable.
  - Fix or mask the fault or failure.
  - Operate in a degraded mode.

Coordination Model

- For each responsibility that must remain highly available:
  - Can coordination mechanisms detect the failure?
    - Is guaranteed delivery of messages necessary?
    - Will coordination work under degraded communication?
  - Can the coordination mechanism replace the failed element?
    - Does replacement of a server allow the system to still operate?
  - Can coordination still function under degraded conditions?
    - How much lost information can the model withstand?

Data Model

- For each element that must be highly available:
  - Which data abstractions, operations, or properties could cause a failure?
  - For each data abstraction, operation, or property, can it be disabled, temporarily unavailable, fixed, or masked?
    - Can write requests be cached if a server is temporarily down, then performed once it is returned to service?

Mapping Among Elements

- What processes, CPUs, persistent storage, etc. can produce a failure?
- Ensure that mapping of elements to runtime elements is flexible enough to permit recovery.
  - Which processes need to be reassigned at runtime?
  - Which CPUs, data stores, etc. can be activated or reassigned at runtime?
  - How can data on failed CPUs or storage be replaced?
  - How quickly can the system be reinstalled?

Resource Management

- What critical resources are needed to continue operating in the presence of a fault?
  - Ensure sufficient resources to log the fault, notify people and systems, disable the events causing the fault, fix or mask the fault, and operate in a degraded mode.
  - Determine availability time for critical resources, what resources must be available at specified time intervals, and repair time.
  - Ensure critical resources are available during these intervals. (i.e., make sure input queues are large enough to buffer if a server fails).

Key Points

- Availability is the ability of the system to be available for use, especially after a failure.
- Failures must be recognized or prevented.
  - System response can range from “ignore it” to “keep on going as if it didn’t occur”.
- Availability tactics detect faults, recover from failures, and prevent failures.
  - Detection depends on signs of life.
  - Recovery involves retrying operations or maintaining redundant data or computations.
  - Prevention depends on removing elements from service or limiting scope of failures.

Next Time

- Embedded System Architecture and Architecture Design Languages
  - Sources:
    - Medvidovic and Taylor, “A classification and comparison framework for Architecture Description Languages”
    - “Architecture Analysis with AADL”

- Homework:
  - Project, Part 3 - Due on Nov 18
  - Reading Assignment 3 - Due on Nov 27